In Felix C. Freiling, Irene Eusgeld, and Ralf Reussner (eds.), Proceedings of the 2008 Dependability Metrics Research Workshop, pages 7–11. University of Mannheim, Department of Computer Science, Technical Report TR-2009-002. May 2009.

Adaptive Capacity Management for the Resource-Efficient Operation of Component-Based Software Systems*

André van Hoorn

Graduate School TrustSoft
University of Oldenburg
D-26111 Oldenburg, Germany

Abstract Overprovisioning capacity management for application service provision causes underutilized computing resources during low or medium workload periods. This paper gives an overview of our work in progress aiming for improving the resource efficiency in operating large component-based software systems that are exposed to highly varying workloads. Based on continuously updated architectural runtime models of the application and its deployment environment, the number of allocated computing resources as well as the deployment of the software components are automatically adapted with respect to current demands and specified performance requirements.

1 Introduction

Today's enterprise applications are complex, business-critical software systems. An important extra-functional characteristic of these systems is performance, consisting of timing behavior and resource utilization [6]. Especially requirements on timing behavior metrics such as throughput or end-to-end response time are part of the so-called Service Level Agreements (SLAs) the provider and the client of a service agreed on. The SLAs constitute a contractual specification regarding the Quality of Service (QoS) that must be satisfied by the application service provider.

Particularly interactive software systems which are accessible through the Internet are exposed to highly varying and bursty workloads, e.g., in terms of the number of concurrent users or the usage profiles [1, 5, 9]. The timing behavior of such systems is significantly influenced by the workload conditions due to resource contention caused by concurrent demands. Over the last years, capacity management for application service provision was performed in a rather static and overprovisioning way, i.e., deploying software components to a fixed infrastructure of application and database servers which satisfy the needs for anticipated peak workload conditions. Future infrastructure demands are satisfied

* This work is supported by the German Research Foundation (DFG), grant GRK 1076/1.

In Felix C. Freiling, Irene Eusgeld, and Ralf Reussner (eds.), Proceedings of the 2008 Dependability Metrics Research Workshop, pages 7–11. University of Mannheim, Department of Computer Science, Technical Report TR-2009-002. May 2009.

in a spirit of "kill-it-with-iron": adding additional resources to the infrastructure or replacing existing resources by more powerful ones. The shortcoming of this approach is that during medium or low workload periods, the allocated resources may be heavily underutilized causing unnecessarily high operating costs due to power consumption or infrastructure leases.

We are working on an automatic approach for adaptive runtime capacity management, overviewed in Section 2, which allows component-based software systems [10] to be operated more efficiently. Efficiency shall be improved by allocating only as much computing resources at a time as required for satisfying the specified SLAs. We consider a set of architecture-level adaptation operations based on which the software system is reconfigured at runtime in a more fine-grained way than for example classic load-balancing or virtualization approaches do.

2 Overview of the Approach

Section 2.1 describes the adaptation operations based on which the software system is reconfigured at runtime. Continuously updated architectural models are used to evaluate the performance of the architecture and that of possible adaptation alternatives. The required information to be captured in the models is outlined in Section 2.2. Section 2.3 gives an overview of the analysis activities.

2.1 Adaptation Operations

We consider the following three architecture-level adaptation operations:

(1) **Node Allocation & Deallocation.** A server node is allocated or deallocated, respectively. In case of an allocation, this includes the installation of an execution environment, e.g., a JBoss runtime environment for Java EE components, but it does not involve any (un)deployment operation of software components. Intuitively, the goal of the allocation is providing additional computing resources and the goal of the deallocation is saving operating costs caused by power consumption or usage fees.

(2) **Software Component Migration.** A software component is undeployed from one execution context and deployed into another. The goals of this fine-grained application-level operation are both to avoid the allocation of additional server nodes or respectively to allow the deallocation of already allocated nodes by executing adaptation operation (1).

(3) **Component-level Load-(un)balancing.** This application-level operation consists of the duplication of a software component and its deployment into another execution context (as well as the reverse direction). Future requests to the component are distributed between the available component instances. The goals of this application-level operation are the same as the goals of operation (2).

In Felix C. Freiling, Irene Eusgeld, and Ralf Reussner (eds.), Proceedings of the 2008 Dependability Metrics Research Workshop, pages 7–11. University of Mannheim, Department of Computer Science, Technical Report TR-2009-002. May 2009.

A middleware layer is responsible for executing the adaptation operations in a way that is transparent to the software system to be adapted. Operation (1) is the most expensive operation in terms of the time required for executing it. In lab experiments, we measured that software component redeployments similar to the adaptation operations (2) and (3) can be executed within milliseconds [4].

2.2 Architectural Models

Relevant static and dynamic aspects of the software architecture are captured in architectural models of the software system. During runtime, these models are updated through measurements, reflect the architectural changes caused by executed adaptation operations, and are used for the continuous analysis (Section 2.3). The following list gives an overview of the information to be captured in the models:

- Components (interfaces and internal performance-relevant behavior)
- Assembly (connection of the components through their interfaces)
- Deployment environment (resources and their performance characteristics)
- Component deployment (mapping of components to execution contexts)
- SLAs and internal performance requirements (component interfaces)
- Adaptation
  - Components to which the adaptation operations are applicable
  - Conditions or rules when to perform an adaptation (analysis)

The performance-relevant modeling of the software architecture will be based on the state of the art in modeling for software performance prediction by annotating architecture models with performance aspects which can be transformed into solvable performance analysis models like queueing networks [2]. Examples for architecture-level software performance modeling notations are the UML SPT/MARTE profiles [7, 8] or the Palladio Component Model [3] for performance prediction of component-based software systems.

2.3 Runtime Analysis

The continuous runtime analysis constitutes the core part of the approach. We identified four main analysis activities to be executed. All these activities rely on the runtime model of the software architecture which needs to be continuously updated through measurements.

(1) **Performance evaluation**. In this activity, the performance of the current system configuration is evaluated. This includes whether or not performance requirements (especially the SLAs) are satisfied and to what degree the resources are utilized.
(2) **Workload analysis and estimation**. The result of this activity is an estimation of the near-future workload derived from trends in past workload measurements.

In Felix C. Freiling, Irene Eusgeld, and Ralf Reussner (eds.), Proceedings of the 2008 Dependability Metrics Research Workshop, pages 7–11. University of Mannheim, Department of Computer Science, Technical Report TR-2009-002. May 2009.

(3) **Performance prediction**. In this activity, the performance of the current system configuration is predicted based on the performance evaluation and the workload estimation in activities (1) and (2). Performance analysis models derived from the architectural models are used for prediction.
(4) **Adaptation analysis**. The effect of possible adaptations on the performance is evaluated using similar techniques as they were used during the performance prediction activity. The result is a selection of adaptation operations to be executed.

3 Conclusions and Future Work

This paper provided on overview of our work in progress on an approach aimed for improving the resource efficiency in operating large component-based software systems which are exposed to highly varying workloads. Based on continuous analyses, the configuration of the software system in terms of the allocated computing resources and the deployment of components to execution contexts, is adapted using three architecture-level adaptation operations.

Since the work is still in an early phase, a lot of future work remains: (1) the adaptation operations will be formally specified and implemented as a proof-of-concept; (2) a suitable notation for the architectural models needs to be identified; and particularly, (3) the required runtime analyses must be developed in detail. We plan to perform an evaluation by simulation in the first place before setting up a case study with a realistic application in the lab.

Acknowledgment

The author thanks the participants of the 2008 DMetrics Workshop, as well as the members of the Software Engineering Group and the Graduate School TrustSoft at the University of Oldenburg for giving valuable early feedback on this work.

References

1. Martin F. Arlitt, Diwakar Krishnamurthy, and Jerry Rolia. Characterizing the scalability of a large web-based shopping system. *ACM Transactions on Internet Technology*, 1(1):44–69, 2001.
2. Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. Model-based performance prediction in software development: A survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, 2004.
3. Steffen Becker, Heiko Koziolek, and Ralf Reussner. The Palladio Component Model for Model-Driven Performance Prediction. *Journal of Systems and Software*, 82(1):3–22, 2009. Special Issue: Software Performance - Modeling and Analysis.
4. Sven Bunge. Transparent redeployment in component-based software systems (in German), December 2008. Diploma Thesis, University of Oldenburg.

In Felix C. Freiling, Irene Eusgeld, and Ralf Reussner (eds.), Proceedings of the 2008 Dependability Metrics Research Workshop, pages 7–11. University of Mannheim, Department of Computer Science, Technical Report TR-2009-002. May 2009.

5. Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, and Alfons Kemper. Workload analysis and demand prediction of enterprise data center applications. In *Proceedings of the 2007 IEEE International Symposium on Workload Characterization (IISWC-2007)*, September 2007.
6. Heiko Koziolek. Introduction to performance metrics. In Irene Eusgeld, Felix Freiling, and Ralf Reussner, editors, *Proceedings of the 2005 Dependability Metrics Workshop (DMetrics)*, volume 4909 of *LNCS*, pages 199–203. Springer, 2008.
7. Object Management Group. UML Profile for Schedulability, Performance, and Time. `http://www.omg.org/cgi-bin/doc?formal/2005-01-02`, January 2005.
8. Object Management Group. UML Profile for Modeling and Analysis of Real-time and Embedded Systems (MARTE), Beta 1. OMG adopted specification ptc/07-08-04. `http://www.omg.org/cgi-bin/apps/doc?ptc/07-08-04.pdf`, August 2007.
9. Christopher Stewart, Terence Kelly, and Alex Zhang. Exploiting nonstationarity for performance prediction. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007 (EuroSys '07)*, pages 31–44. ACM, 2007.
10. Clemens Szyperski. *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, 2. edition, 2002.